Computergestützte Mathematik zur Linearen Algebra

Programmsteuerung II

Achim Schädle

Übungsleiter: Lennart Jansen

Tutoren: Marina Fischer, Kerstin Ignatzy, Narin Konar
Pascal Kuhn, Nils Sänger, Tran Dinh

20. November 2014

Steuerung von Programmabläufen

MATLAB-Skripte und Funktionen enthalten Abfolgen von Befehlen, die Zeile für Zeile ausgewertet werden.

Fragen

- Wie sorge ich dafür, dass bestimmte Befehle nur unter bestimmten Bedingungen ausgewertet werden?

  Fallunterscheidungen

- Ist es möglich sich wiederholende Befehle zusammenzufassen oder sogar sooft zu wiederholen, bis eine Bedingung erfüllt ist?

  Schleifen

Bedingungen — Logische Ausdrücke

Typische Kontrollstrukturen benötigen Vergleiche, wie z.B.

$$a = \begin{cases} b & \text{wenn } b > c \\ c & \text{sonst} \end{cases}$$

Vergleichsoperatoren (komponentenweise) (vgl. VL 23.10 & 30.10)

MATLAB-Syntax	math. Ausdruck
==	$=$
~=	$\neq$
<	$<$
>	$>$
<=	$\leq$
>=	$\geq$

- Vergleiche liefern entweder wahr (1) oder falsch (0)

Bedingungen — Logische Operatoren

Logische Ausdrücke können miteinander verknüpft werden, um komplexere Fälle zu definieren

$$a = \begin{cases} b & \text{wenn } b > c \text{ und } c > 0 \\ c & \text{sonst} \end{cases}$$

Logikoperatoren

MATLAB-Syntax	log. Ausdruck
`&&`	und
`||`	oder
`not( )`	nicht

- Jeder nicht-Null Zahlenwert wird als wahr ausgewertet nur Null entspricht falsch
- Mit Hilfe von `logical`, `true` und `false` können logische Variablen angelegt werden

Fallunterscheidungen I: `if`-Konstrukte

Sollen **MATLAB**-Befehle nur unter bestimmten Bedingungen ausgeführt werden, verwendet man Fallunterscheidungen

Einfachste Form

```
if Bedingung
   Matlab-Befehle
end
```

- Die **MATLAB**-Befehle werden nur ausgeführt, wenn `Bedingung` wahr ist, sonst übersprungen
- Die **MATLAB**-Befehle werden durch die Schüsselwörter `if` und `end` eingeschlossen

Fallunterscheidungen II: `if-else`-Konstrukte

Mit Hilfe des Schüsselwortes `else` kann das `if`-Konstrukt um alternative **MATLAB**-Befehle ergänzt werden

Zwei Alternativen

```
if Bedingung
   Matlab-Befehle % (falls Bedingung wahr)
else
   Matlab-Befehle % (falls Bedingung falsch)
end
```

- Es wird unabhängig davon ob die `Bedingung` wahr oder falsch ist, ein Satz von **MATLAB**-Befehlen ausgeführt
- `end` signalisiert wieder das Ende des `if-else`-Konstrukts

Fallunterscheidungen III: `if-elseif`-Konstrukte

Drei oder mehr Alternativen

```
if Bedingung 1
   Matlab-Befehle % (falls Bedingung 1 wahr)
elseif Bedingung 2
   Matlab-Befehle % (falls Bedingung 1 falsch
                  %   und Bedingung 2 wahr)
elseif Bedingung 3
   ...
else               % optional
   Matlab-Befehle % (falls alle Bedingungen
                  %          vorher falsch)
end
```

- Es wird nur ein Block von **MATLAB**-Befehlen ausgewertet

Fallunterscheidungen IV: Beispiel

Berechnung der skalaren Funktion

$$f(x) = \begin{cases} -2 & x \leq -2 \\ x^2 - 2 & x > -2 \text{ und } x < 2 \\ 2 & x \geq 2 \end{cases}$$

```
function f = piecewisefunction(x)

if x <= -2
   f = -2;
elseif x < 2        % x > -2, wenn x <= -2 nicht wahr
   f = x^2 - 2;
else                % x > 2, wenn x < 2 nicht wahr
   f = 2;
end
```

Schleifen

Bisher haben wir **MATLAB**-Befehle nacheinander im Command Window eingegeben oder in einer `m`-Datei gespeichert

- **MATLAB** arbeitet die Befehle nacheinander ab
- Mehrfach auszuführende Befehle, müssen mehrfach eingegeben werden

Schleifen (Zusammenfassung sich wiederholender Befehle)

- Grundlegende Unterscheidung in zwei Arten

for-Schleifen

```
for var = mat
   Matlab-Befehle
end
```

while-Schleifen

```
while Bedingung
   Matlab-Befehle
end
```

for-Schleifen

Allgemeine Form siehe Vorlesung vom 13.11

```
for var = mat
   Matlab-Befehle
end
```

- Im i-ten Durchlauf der Schleife gilt `var = mat(:,i)`, d.h. `var` enthält die i-te Spalte der Matrix `mat`
- Die Schleife wird wiederholt, bis alle Spalten von `mat` durchlaufen wurden (d.h., `size(mat,2)`-mal)
- Es muss vorher bekannt sein, wieoft die Befehle wiederholt werden und u. U. welche Werte `var` in jedem Durchlauf enthalten soll

Häufig

- `mat = 1:n` (`var = i` im i-ten Durchlauf)
- `mat = anfang:inkrenent:ende`
- `mat = linspace(a,b,N)`

while-Schleifen

Allgemeine Form

```
while Bedingung
   Matlab-Befehle
end
```

- Bedingung ist ein logischer Ausdruck
- Die Schleife wird wiederholt, solange Bedingung wahr ist
- Man muss nicht vorher wissen, wieoft die Befehle wiederholt werden sollen

Achtung

- Man muss sicherstellen, dass die Bedingung irgendwann falsch wird, da ansonsten die Schleife endlos[1] läuft

[1] Abbruch mit Ctrl-c

while-Schleifen: Beispiel I

Aufgabe: Finde den größten gemeinsamen Teiler von $m, n \in \mathbb{N}$ mit Hilfe des euklidischen Algorithmus

```
function g = ggT(m,n)

while m ~= n
   if m > n
      m = m - n;
   else
      n = n - m;
   end
end
g = n;
```

while-Schleifen: Beispiel II

Aufgabe: Finde zu gegebenem $q \in (0,1)$ und Toleranz δ ab welchem $n \in \mathbb{N}$ gilt

$$\left| \sum_{k=0}^{n} q^k - \frac{1}{1-q} \right| < \delta$$

```
function n = geoapprox(q,delta)

sum = 0;
k = 0;
while abs(sum - 1/(1-q)) > delta
    sum = sum + q^k;
    k = k + 1;
end
n = k
```

Rekursion

Mit Hilfe von Kontrollkonstrukten können auch rekursive[2] Funktionen realisiert werden.

Beispiel: Berechnung der Fibonacchi-Folge

$$F_0 = 1, F_1 = 1, F_n = F_{n-1} + F_{n-2}, n \geq 2$$

```
function F = fibonacchi(n)

if n == 0 || n == 1
   F = 1;
else
   F = fibonacchi(n-1) + fibonacchi(n-2);
end
```

Es muss sichergestellt werden, dass Rekursionen abbrechen!

[2] Funktionen, die sich selbst aufrufen

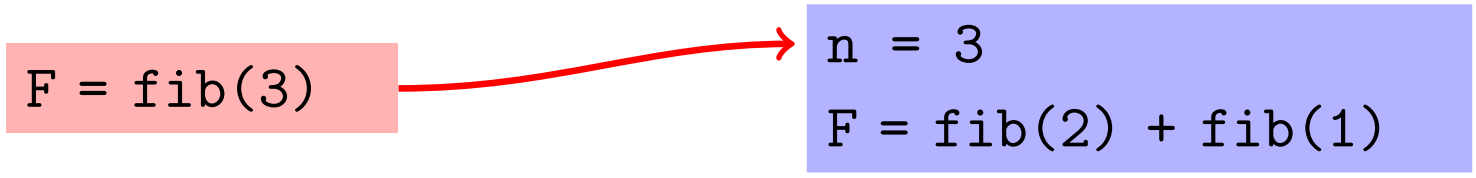
Bearbeitung von `fib(3)`

MATLAB workspace

```
F = fib(3)
```

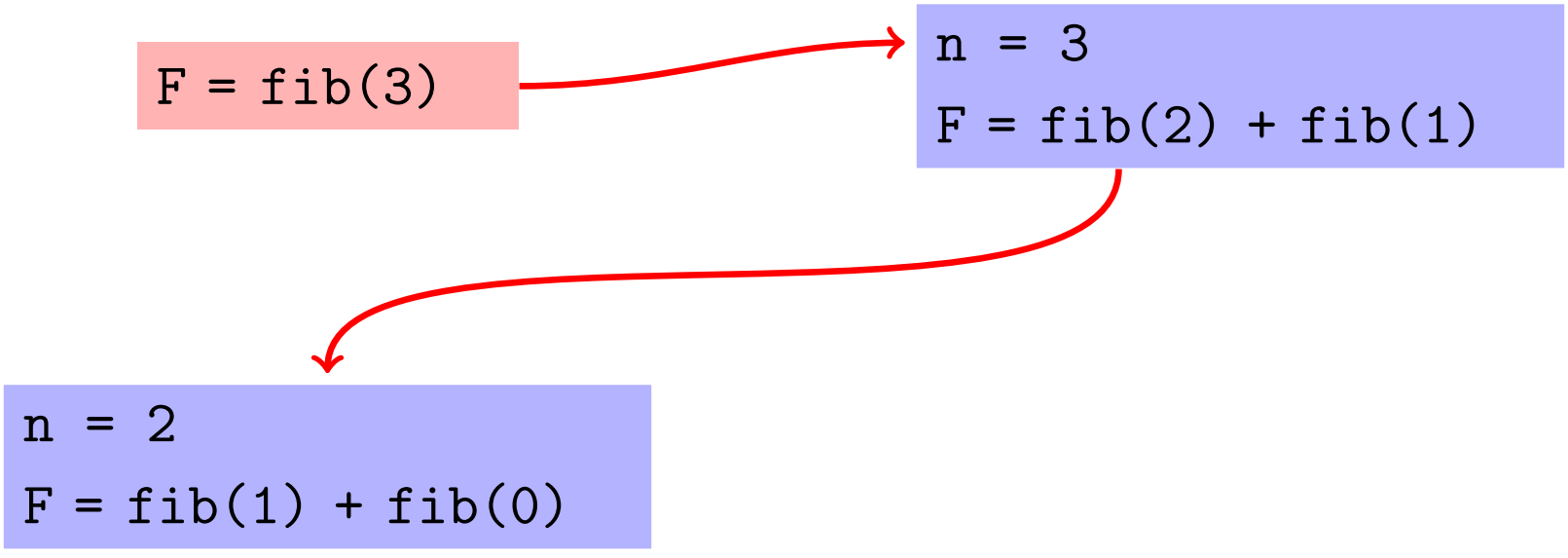
Bearbeitung von `fib(3)`

MATLAB workspace

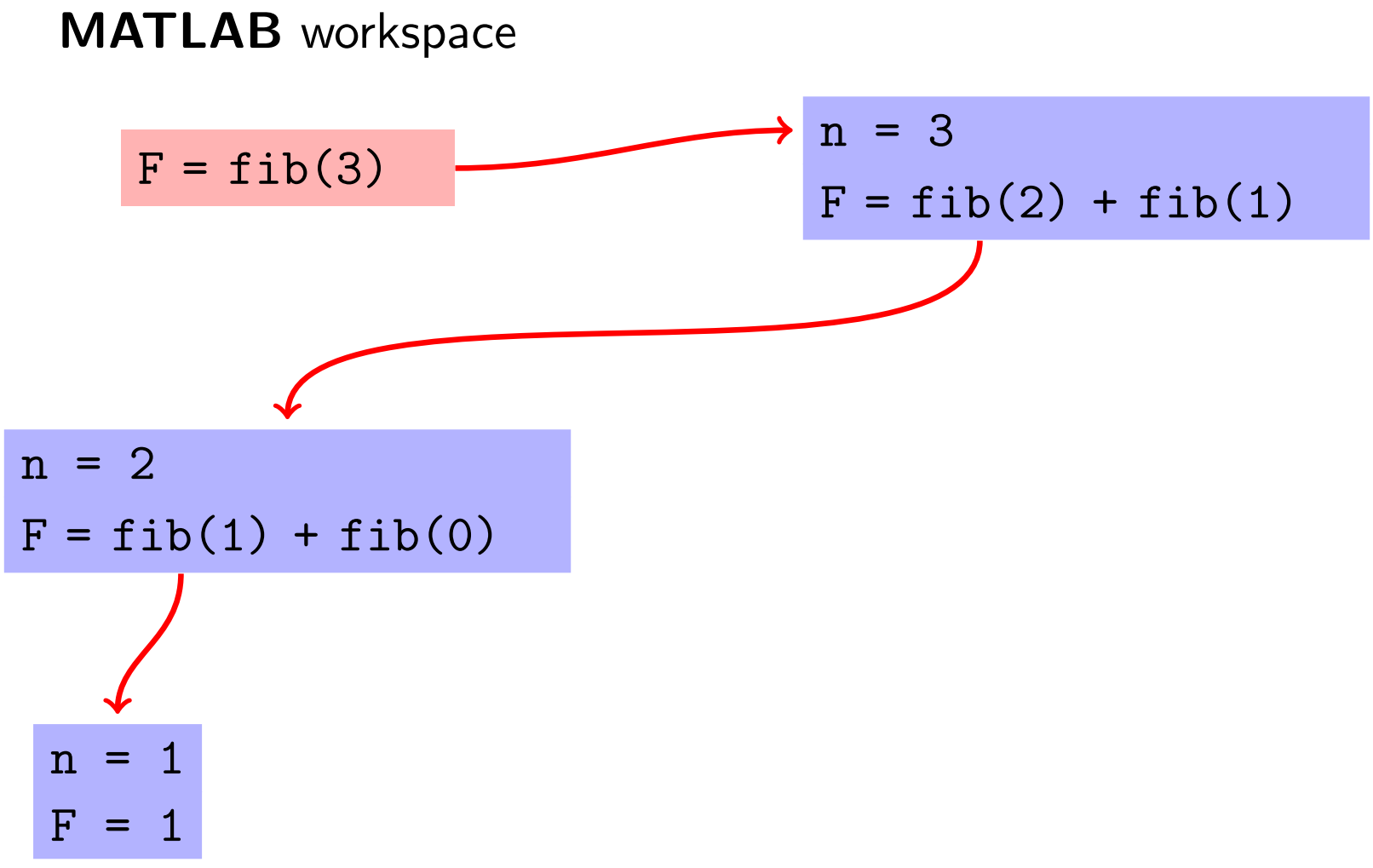


Bearbeitung von `fib(3)`

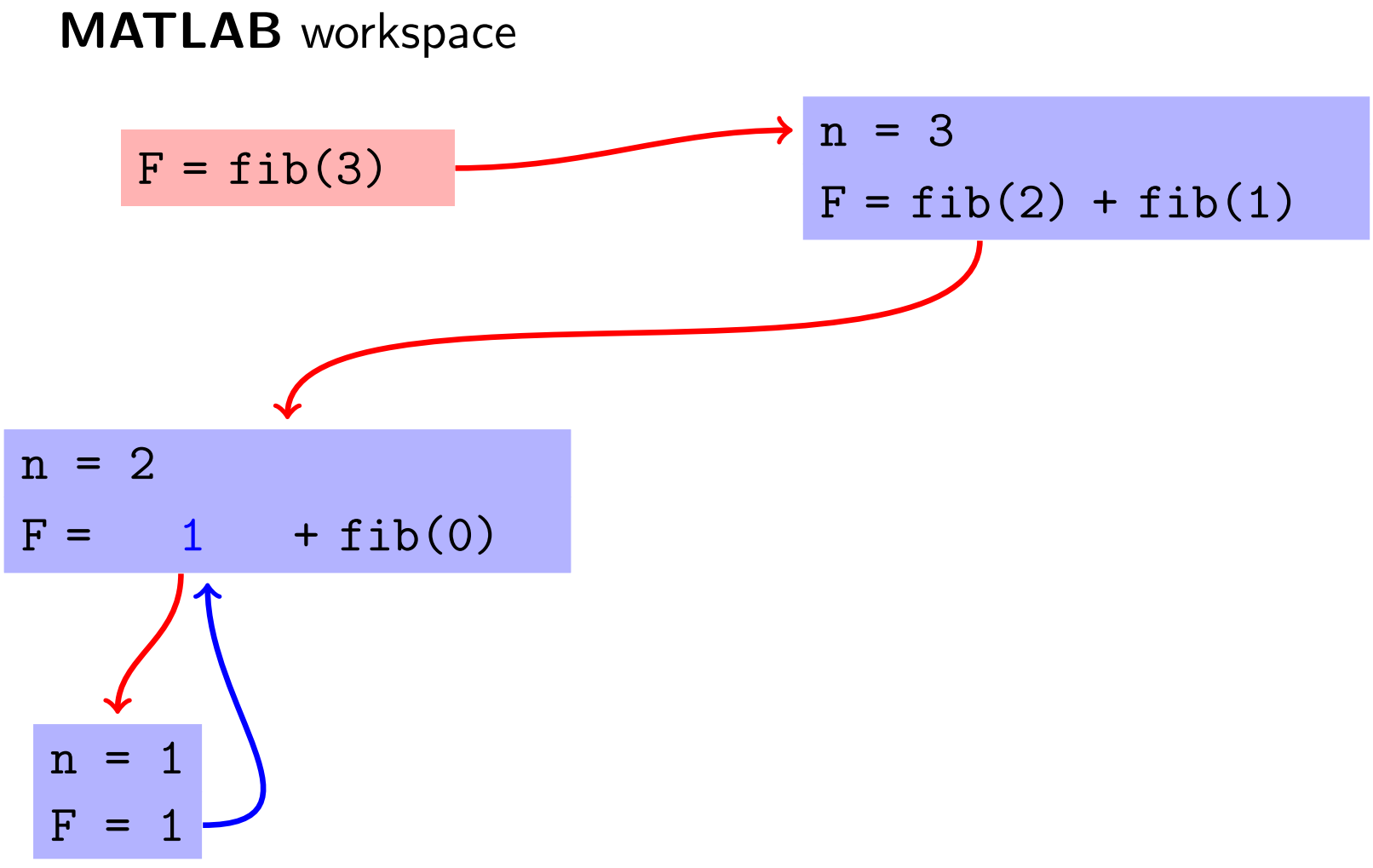
MATLAB workspace



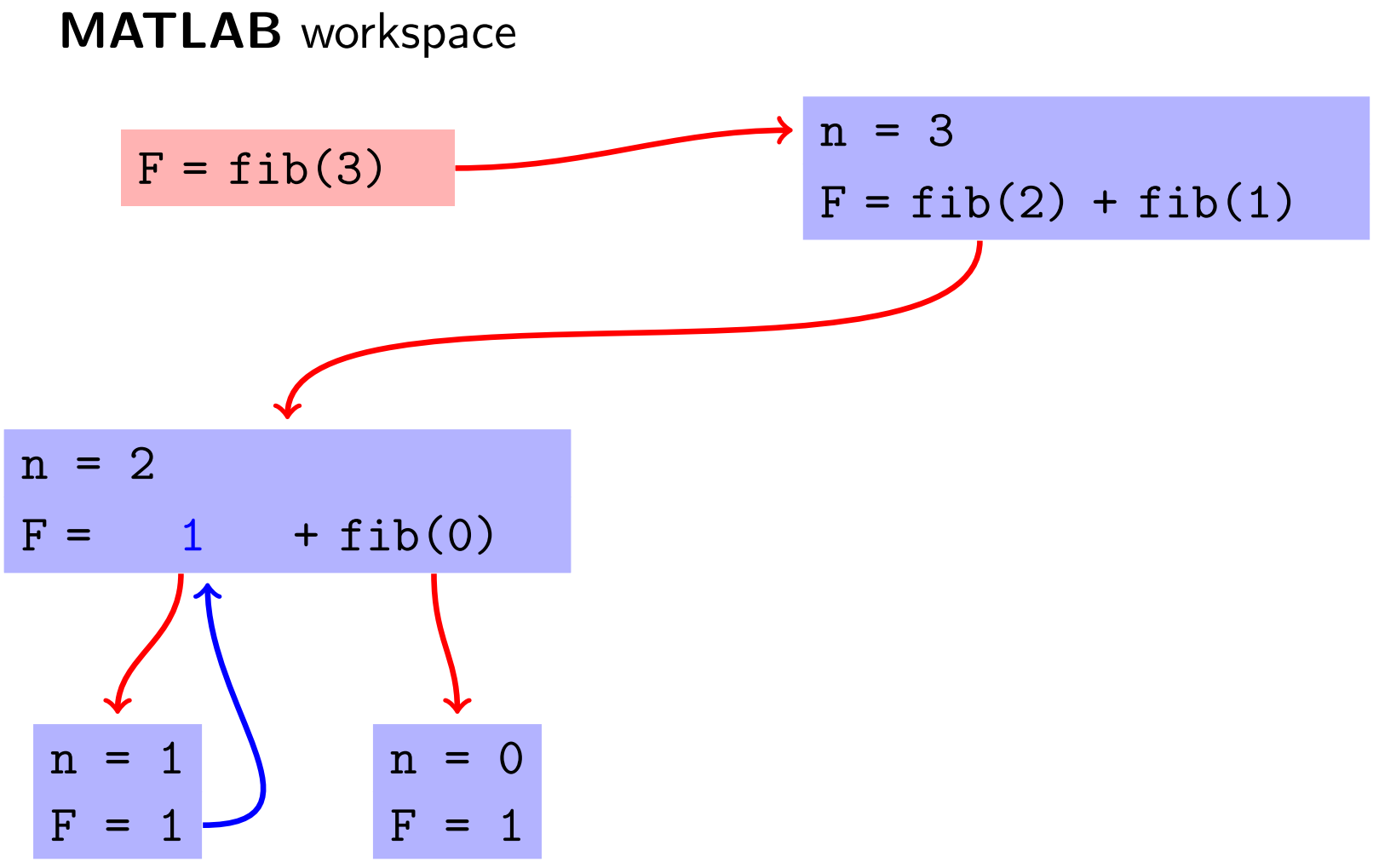
Bearbeitung von `fib(3)`



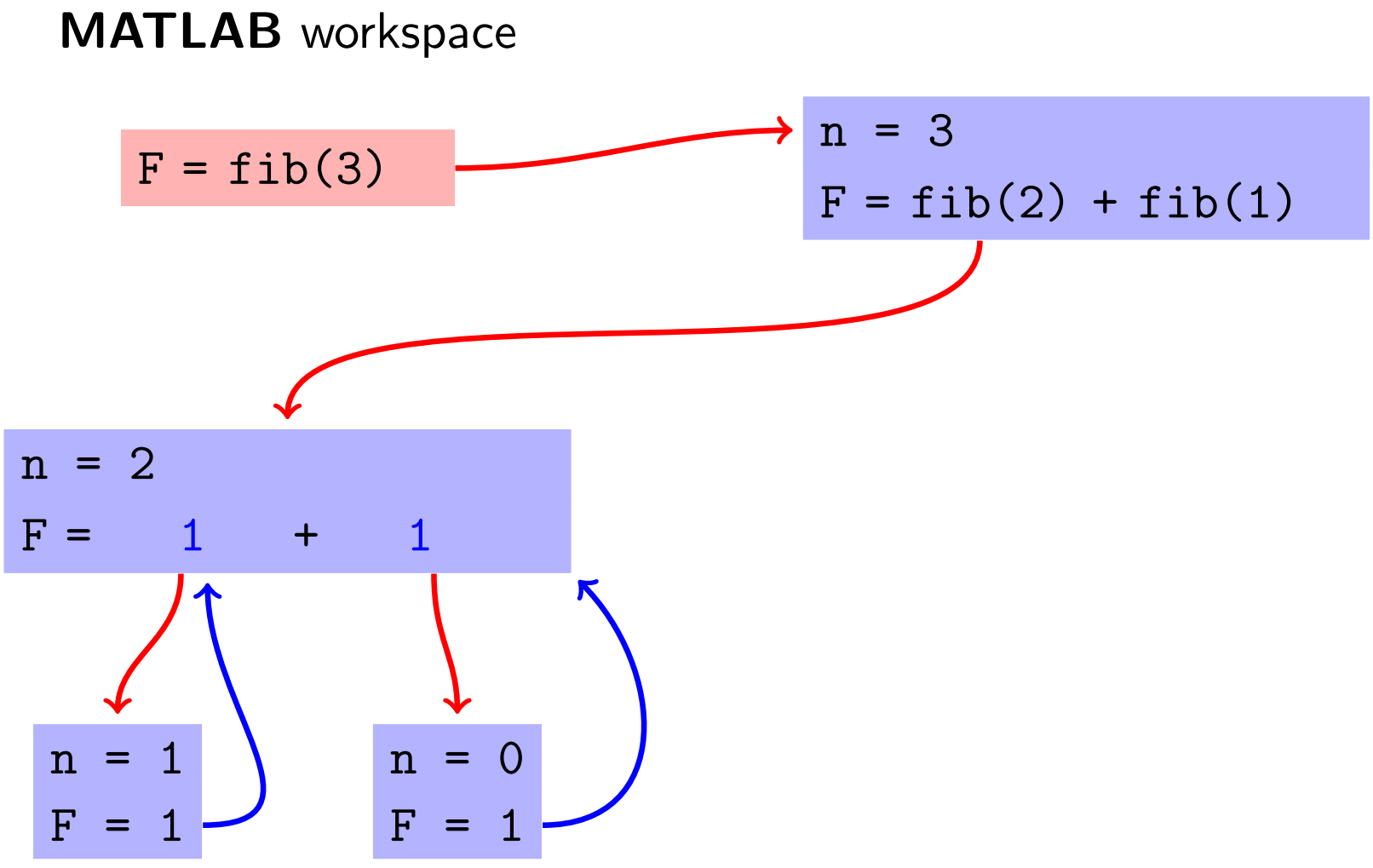
Bearbeitung von `fib(3)`



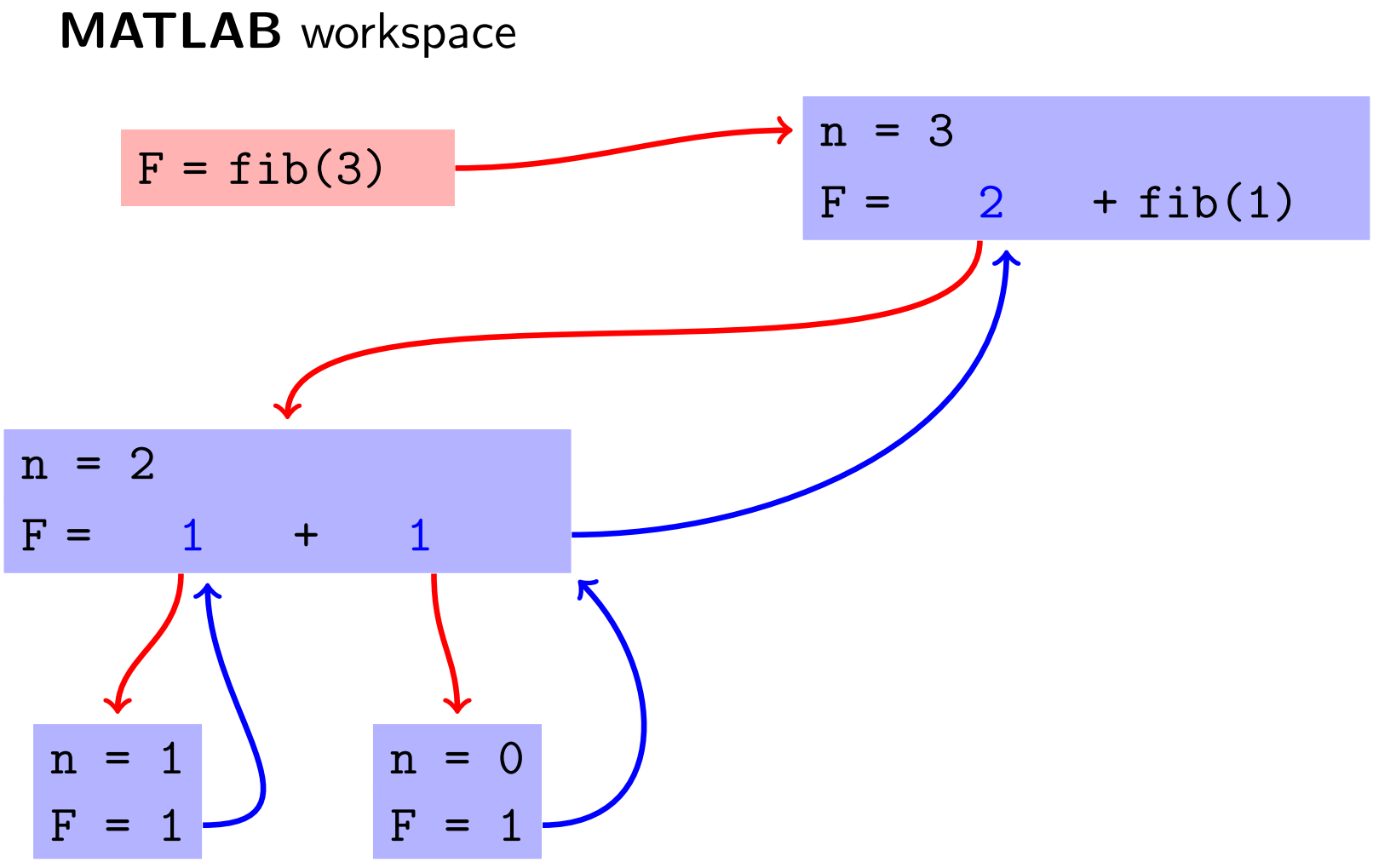
Bearbeitung von `fib(3)`



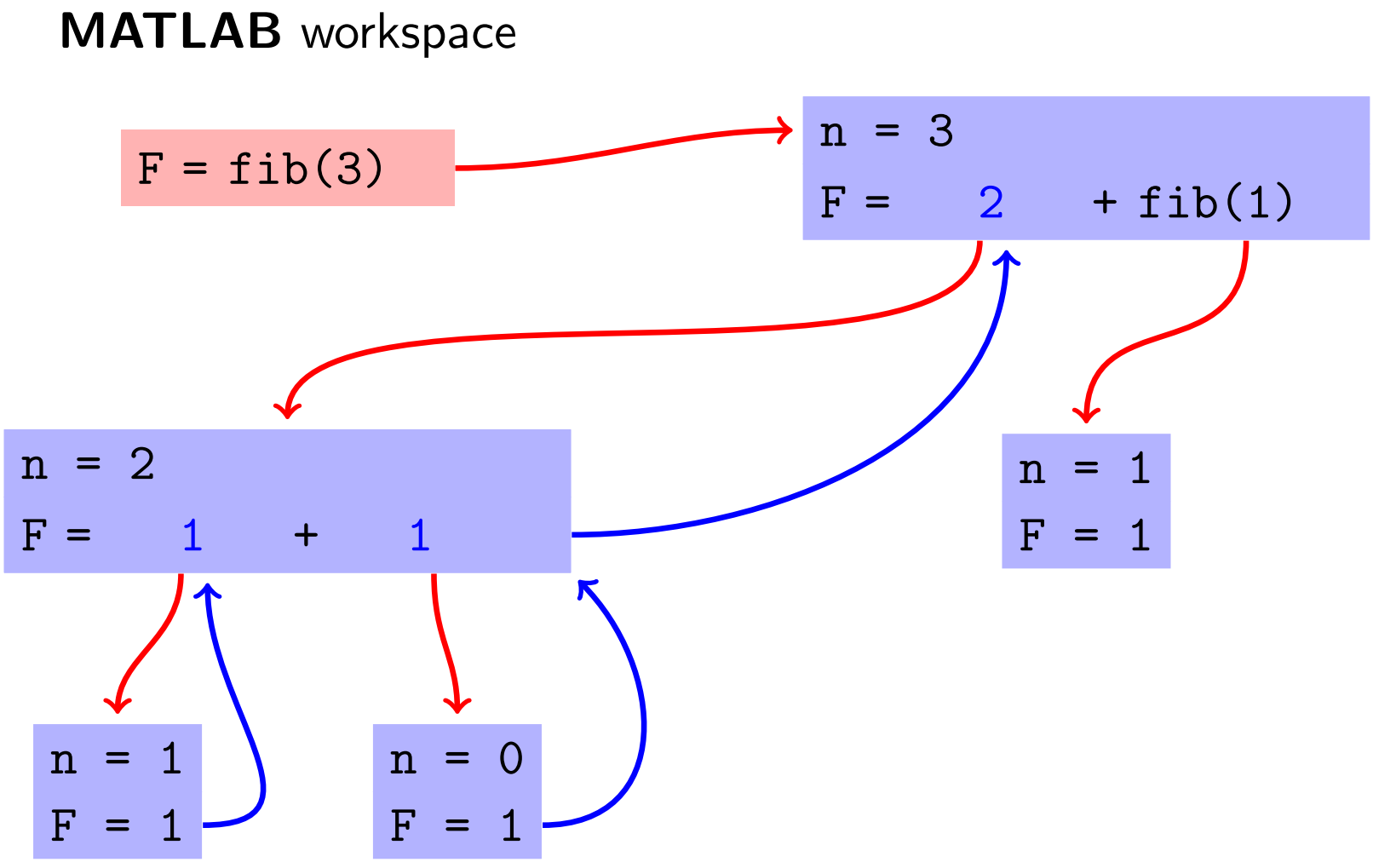
Bearbeitung von `fib(3)`



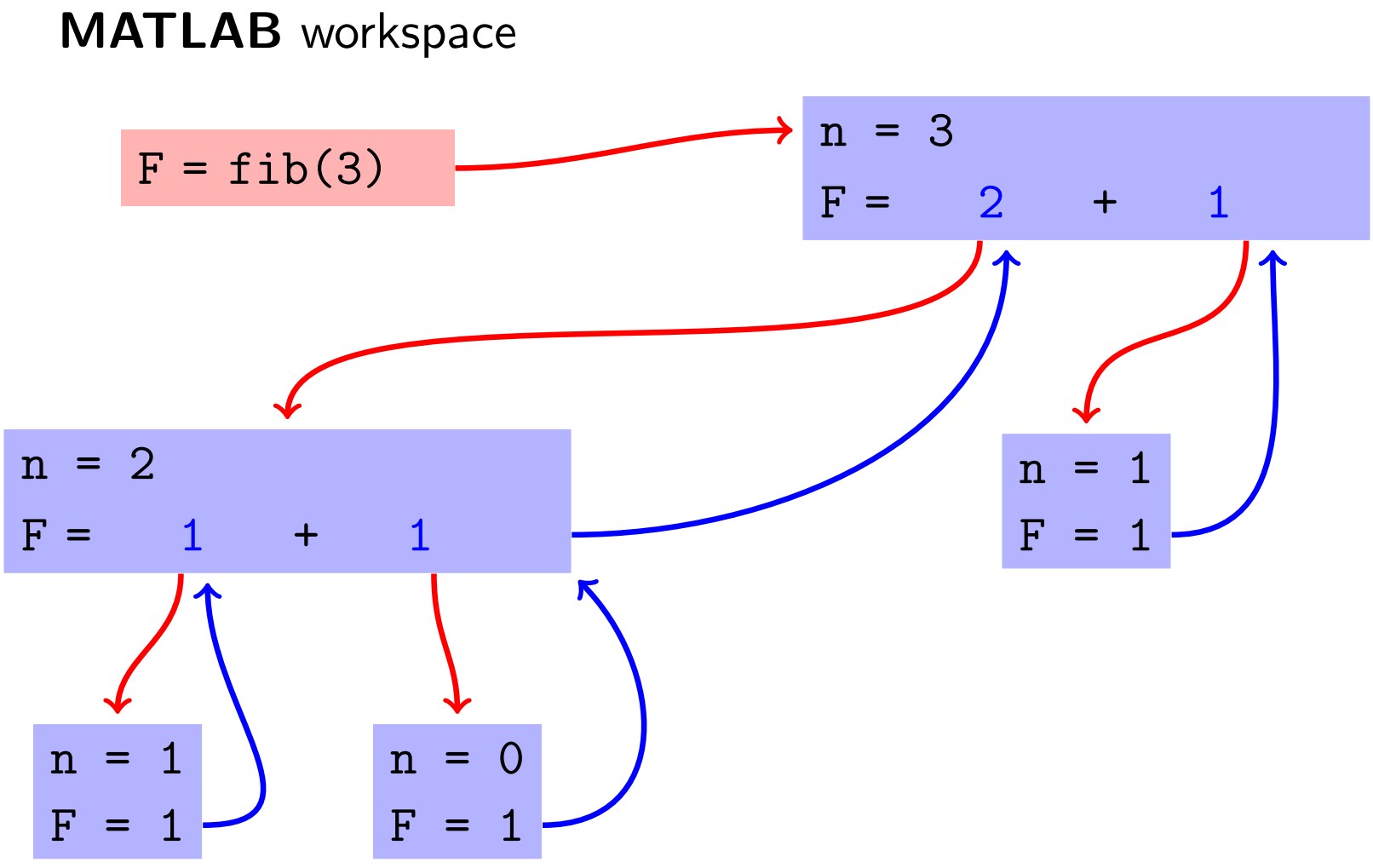
Bearbeitung von `fib(3)`



Bearbeitung von `fib(3)`



Bearbeitung von `fib(3)`



Bearbeitung von `fib(3)`